Project 3 - Building Networks with Tensorflow and Keras

Anna-Maria Nau (anau) and Christoph Metzner (cmetzner)
COSC 525 - Deep Learning
University of Tennessee, Knoxville

March 2020

1 Introduction

This project aims to learn how to build deep learning network architectures using Tensorflow and Keras library for Python 3. Therefore, the following five different network architectures were built: 1) a fully connected neural network, 2) a small and 3) a big convolutional neural network (CNN), 4) a modified CNN, and 5) a variational autoencoder.
The Fashion-MNIST dataset was used to compare the performance of all networks in terms of classification accuracy, loss reduction over epochs, as well as computation time needed. The Fashion-MNIST dataset consists of a training dataset with 60,000 and a testing dataset with 10,000 samples of ten different classes of clothes. Furthermore, the image format was 28x28 pixels and grey-scaled. The ten classes were t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. The training and testing datasets were scaled using the minimal (0) and maximal (255) values of the training dataset.

Figure 1: First 20 samples of Fashion-MNIST training dataset

2 Introduction of Neural Networks

2.1 Task 1: Fully Connected Neural Network

This fully connected neural network consists of several fully connected hidden layers and an output layer. The first hidden layer uses 784 neurons and the hyperbolic tangent activation function to transform the output. The second hidden layer uses the sigmoid activation function in each of the 512 neurons, while the third hidden layer consists of 100 neurons and uses the rectified linear activation function. Ten output neurons are used using the softmax activation function to represent the output.

```
Model: "Fully_Connected_NN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
FC_Input_Layer (Flatten)     (None, 784)               0
_________________________________________________________________
FC_Hidden_Layer_1 (Dense)    (None, 784)               615440
_________________________________________________________________
FC_Hidden_Layer_2 (Dense)    (None, 512)               401920
_________________________________________________________________
FC_Hidden_Layer_3 (Dense)    (None, 100)               51300
_________________________________________________________________
FC_Output_Layer (Dense)      (None, 10)                1010
=================================================================
Total params: 1,069,670
Trainable params: 1,069,670
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 2: Summary of fully connected neural network.

2.2 Task 2: Small Convolutional Neural Network

This network utilizes a convolutional layer with 40 filters with size 5x5, stride one, and no padding to detect relevant features of each grey-scaled image. The output is transformed using the rectified linear activation function. A 2x2 max-pooling layer is used to concentrate on the most relevant output and reduce the dimension, which reduces computational effort. The flattened output is connected with a fully connected layer with 100 hidden neurons with the rectified linear activation function. Ten output neurons are used using the softmax activation function to represent the output.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 40)        1040
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 40)        0
_________________________________________________________________
flatten (Flatten)            (None, 5760)              0
_________________________________________________________________
dense (Dense)                (None, 100)               576100
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 578,150
Trainable params: 578,150
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Summary of small convolutional neural network.

2.3 Task 3: Big Convolutional Neural Network

This network utilizes a convolutional layer with 48 filters with size 3x3, stride one, and no padding to detect relevant features of each grey-scaled image. The output is transformed using the rectified linear activation function. A 2x2 max-pooling layer is used to concentrate on the most relevant output and reduce the dimension, which reduces computational effort. Another convolution layer is used to further identify relevant features, using 96 feature detectors with size 3x3, rectified linear activation function, stride of 1, and no padding. A second 2x2 max-pooling layer is used to concentrate on the most relevant output and reduce the dimension of the output. The generated flattened output is connected with a fully connected layer with 100 hidden neurons with the rectified linear activation function. Ten output neurons are used using the softmax activation function to represent the output.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 48)        480
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 48)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 96)        41568
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 96)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2400)              0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               240100
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
=================================================================
Total params: 283,158
Trainable params: 283,158
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 4: Summary of big convolutional neural network.

2.4 Task 4: Modified Convolutional Neural Network

This network utilizes a convolutional layer with 30 filters with size 3x3, stride one, and no padding. The output is transformed using the rectified linear activation function. A 2x2 max-pooling layer is used to concentrate on the most relevant output, reducing its dimensionality. To prevent overfitting, a dropout layer with a rate of 0.2 is used. The generated flattened output is connected with a fully connected layer containing 100 hidden neurons with the rectified linear activation function. Ten output neurons are used using the softmax activation function to represent the output.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 30)        300
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 30)        0
_________________________________________________________________
dropout (Dropout)            (None, 13, 13, 30)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 5070)              0
_________________________________________________________________
dense_4 (Dense)              (None, 100)               507100
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1010
=================================================================
Total params: 508,410
Trainable params: 508,410
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 5: Summary of own convolutional neural network.

2.5 Task 5: Variational Autoencoder

Two variational autoencoders were built - one with latent dimension 10 (Figure 6) and one with latent dimension 32 (Figure 7). The first layer of each encoder is the input layer with shape (28, 28, 1). Next up, the models contain two 2D convolutional layers with 32 and 64 filters, respectively, of size 3x3, stride of 2 and 'same' padding using the rectified linear activation function. Then the data from the convolutional layer is flattened into one-dimensional space before being connected to a dense layer with 16 output neurons using the rectified linear activation function. The next two layers are both linked to the previous dense layer and output the mean and the standard deviation values of the encoded input allowing the encoder model to sample the random variables that constitute the point in the latent space onto which some input is mapped.
The decoder models of each network also begin with an input layer of shape (10,) (Figure 6b) and (32,) (Figure 7b) which is the vector sampled with the encoder model. Next up, the decoder models contains two 2D transpose convolutional layers with 64 and 32 filters, respectively, of size 3x3, stride of 2 and 'same' padding using the rectified linear activation function and lastly, one 2D transpose convolutional layer with 1 filter of size 3x3, stride of 1 and 'same' padding using the sigmoid activation function in order to transform the data from input shape (latent dimension,) to output shape (28, 28, 1).

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 28, 28, 1)	0	
conv2d_1 (Conv2D)	(None, 14, 14, 32)	320	encoder_input[0][0]
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18496	conv2d_1[0][0]
flatten_1 (Flatten)	(None, 3136)	0	conv2d_2[0][0]
dense_1 (Dense)	(None, 16)	50192	flatten_1[0][0]
z_mean (Dense)	(None, 10)	170	dense_1[0][0]
z_log_var (Dense)	(None, 10)	170	dense_1[0][0]
z (Lambda)	(None, 10)	0	z_mean[0][0] z_log_var[0][0]

Total params: 69,348
Trainable params: 69,348
Non-trainable params: 0

(a)

Layer (type)	Output Shape	Param #
z_sampling (InputLayer)	(None, 10)	0
dense_2 (Dense)	(None, 3136)	34496
reshape_1 (Reshape)	(None, 7, 7, 64)	0
conv2d_transpose_1 (Conv2DTr	(None, 14, 14, 64)	36928
conv2d_transpose_2 (Conv2DTr	(None, 28, 28, 32)	18464
decoder_output (Conv2DTransp	(None, 28, 28, 1)	289

Total params: 90,177
Trainable params: 90,177
Non-trainable params: 0

(b)

Figure 6: Summary of variational autoencoder with latent dimension 10: a) encoder model b) decoder model.

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	(None, 28, 28, 1)	0	
conv2d_3 (Conv2D)	(None, 14, 14, 32)	320	encoder_input[0][0]
conv2d_4 (Conv2D)	(None, 7, 7, 64)	18496	conv2d_3[0][0]
flatten_2 (Flatten)	(None, 3136)	0	conv2d_4[0][0]
dense_3 (Dense)	(None, 16)	50192	flatten_2[0][0]
z_mean (Dense)	(None, 32)	544	dense_3[0][0]
z_log_var (Dense)	(None, 32)	544	dense_3[0][0]
z (Lambda)	(None, 32)	0	z_mean[0][0] z_log_var[0][0]

Total params: 70,096
Trainable params: 70,096
Non-trainable params: 0

(a)

Layer (type)	Output Shape	Param #
z_sampling (InputLayer)	(None, 32)	0
dense_4 (Dense)	(None, 3136)	103488
reshape_2 (Reshape)	(None, 7, 7, 64)	0
conv2d_transpose_3 (Conv2DTr	(None, 14, 14, 64)	36928
conv2d_transpose_4 (Conv2DTr	(None, 28, 28, 32)	18464
decoder_output (Conv2DTransp	(None, 28, 28, 1)	289

Total params: 159,169
Trainable params: 159,169
Non-trainable params: 0

(b)

Figure 7: Summary of variational autoencoder with latent dimension 32: a) encoder model b) decoder model.

3 Results

All networks were tested under the same conditions. Mini-batch stochastic gradient descent (SGD) was used as the optimizer with a batch size of 200 samples. The categorical cross-entropy loss was used for the fully connected and convolutional neural networks and the mean-squared error loss for the variational autoencoder to determine the loss after each epoch. The performance of all networks is compared after 50 epochs of training. Furthermore, the networks are evaluated by comparing their performance using the unseen test dataset.

3.1 Task 1: Fully Connected Neural Network

The fully connected neural network classified 86.17% of the training images correctly after the final training epoch, 50, which ultimately resulted in a categorical cross-entropy loss of 0.3897. Slightly lower classification accuracy was achieved for the testing data set with 84.61% and a cross-entropy loss of 0.4163. Figure 8 shows different performance plots a) model loss vs epochs, b) model loss vs. time, and c) model accuracy vs epochs. The greatest drop of the loss takes place in the early stages of the training process as shown in Figure 8a) approximately epoch 10 and 8b) 75 seconds. Figure 8c) indicates an increasing trend of accuracy with increasing numbers of training runs (i.e., epochs). However, the positive return of increasing accuracy reduces with additional training runs.

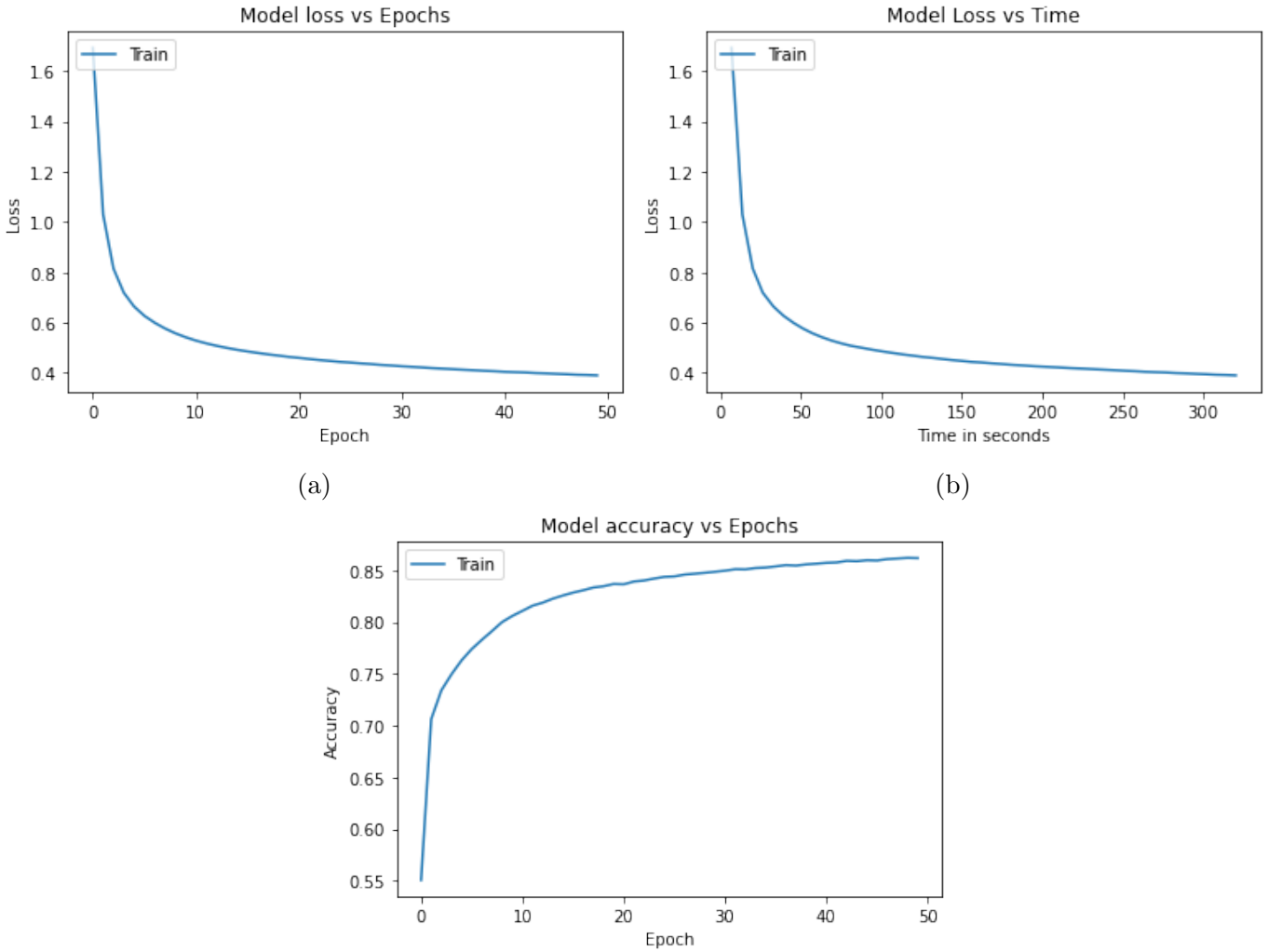


Figure 8: Performance plots for fully connected neural network: a) epoch-loss plot, b) time-loss plot, and c) epoch-accuracy plot.

The confusion matrix (Figure 9) shows the classification performance of the fully connected neural network for the test data set, where the rows represent the true class labels and the columns refer to the predicted labels. The classification accuracy for most classes lies above 80% (1000 samples per class), with the best performance for the class *trouser* (95.3%). Whereas, the lowest accuracy is achieved for the class *shirt* with 54.4%. The results further indicate, that specific types of clothes are more similar to each other than to other types of clothes. For example, the class *shirt* shows rather higher similarities between items from the class *T-shirt/top* (152 wrong assignments), *pullover* (127), or *coat* (117) as the wrongly assigned labels are high. Interestingly, the model easily differentiates between classes related to any type of shoes (*ankle boot*, *sneaker*, and *sandals*) and all other classes, as the wrongly assigned labels, are mostly contributed to the shoe-related classes.

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	845	2	15	37	6	1	79	0	15	0
Trouser	5	953	6	26	5	0	3	0	2	0
Pullover	21	3	740	10	154	1	63	0	8	0
Dress	33	12	16	857	41	0	37	0	4	0
Coat	0	1	100	30	800	0	64	0	5	0
Sandal	0	0	0	1	0	910	0	57	4	28
Shirt	152	1	127	32	117	0	544	0	27	0
Sneaker	0	0	0	0	0	34	0	927	0	39
Bag	2	1	10	11	3	3	15	5	950	0
Ankle boot	0	0	0	0	0	17	0	47	1	935

Figure 9: Confusion matrix of testing data for fully connected neural network with true labels (rows) and predicted labels (columns).

3.2 Task 2: Small Convolutional Neural Network

The small convolutional neural network identified 89.74% of the training images correctly after the final training epoch, 50, which ultimately resulted in a categorical cross-entropy loss of 0.2865. Slightly lower classification accuracy was achieved for the testing dataset with 88.59% and a loss of 0.3225. Figure 10 presents different performance plots a) model loss vs epochs, b) model loss vs. time, and c) model accuracy vs epochs. The greatest drop of the loss takes place in the early stages of the training process as shown in Figure 10a) approximately epoch 10 and 10b) 25 seconds. Figure 10c) indicates an increasing trend of accuracy with increasing numbers of training runs (i.e., epochs). However, the positive return of increasing accuracy reduces with additional training runs.

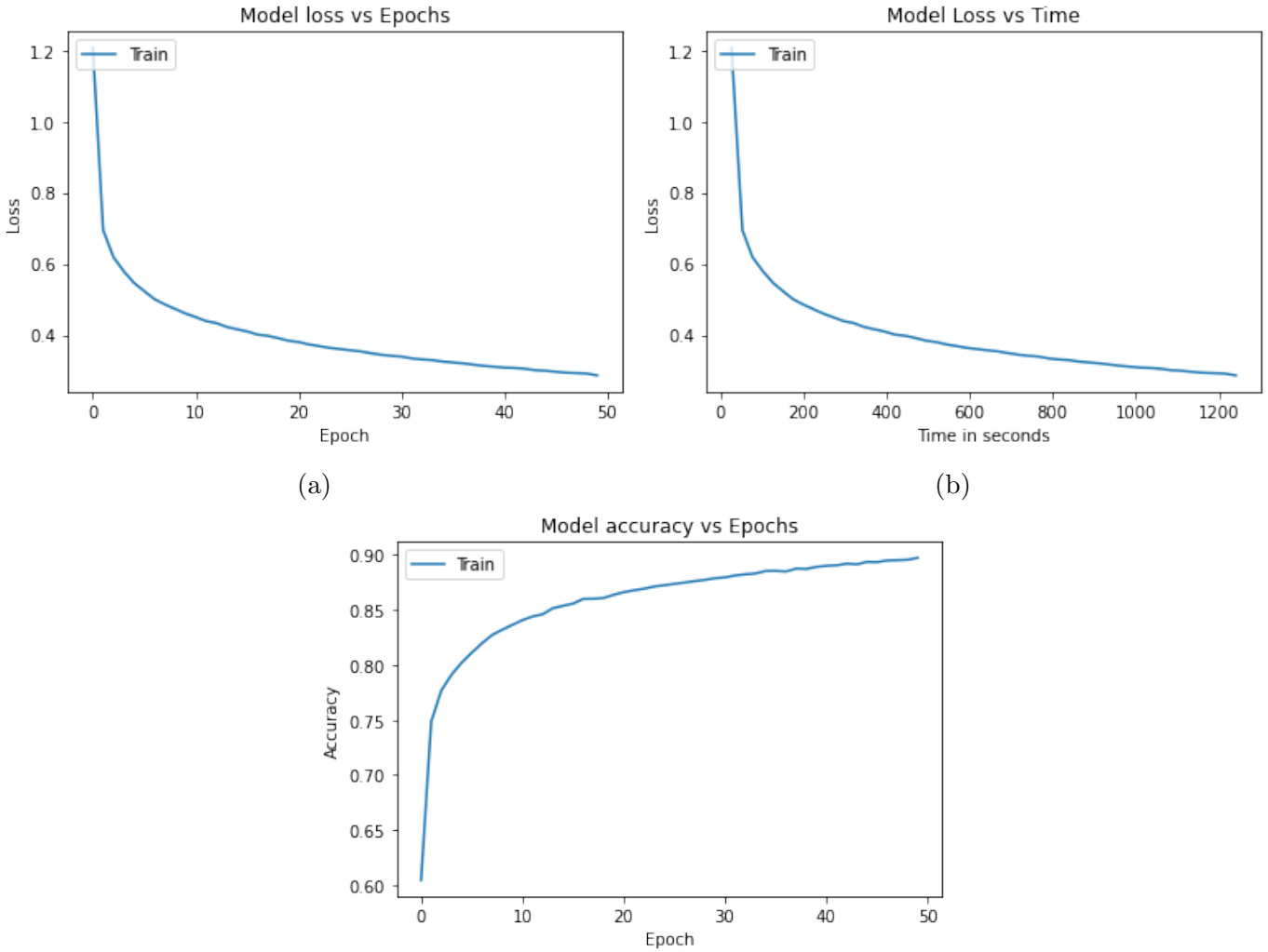


Figure 10: Performance plots for the small convolutional neural network: a) epoch-loss plot, b) time-loss plot, and c) epoch-accuracy plot.

The confusion matrix (Figure 11) shows the classification performance of the small convolutional neural network for the test data set, where the rows represent the true class labels and the columns refer to the predicted labels. The classification accuracy for most classes lies above 85% (1000 samples per class), with the best performance for the class *trouser* (96.8%) and *Sandal* (96.8%), and the lowest for the class *Shirt*. The results further indicate that specific types of clothes are more similar to each other than to other types of clothes. For example, the class *Shirt* shows rather higher similarities between items from the class *T-shirt/Top* with 174 wrongly assigned labels and *pullover* with 114 false predictions. The model can differentiate between classes related to any type of shoes (*ankle boot*, *sneaker*, and *sandals*) and all other classes, as the wrongly assigned labels, are mostly contributed to the shoe-related classes.

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	887	0	26	21	4	1	54	0	7	0
Trouser	3	968	1	19	2	1	4	0	2	0
Pullover	18	1	866	11	57	1	42	0	4	0
Dress	24	7	9	899	33	0	25	0	3	0
Coat	0	1	103	29	803	0	62	0	2	0
Sandal	0	0	0	1	0	968	0	17	1	13
Shirt	174	1	114	21	67	0	612	0	11	0
Sneaker	0	0	0	0	0	32	0	931	1	36
Bag	5	0	12	6	2	1	3	6	965	0
Ankle boot	0	0	0	0	0	7	1	32	0	960

Figure 11: Confusion matrix for the small convolutional neural network.

3.3 Task 3: Big Convolutional Neural Network

The big convolutional neural network classified 88.95% of the training images correctly after the final training epoch, 50, which ultimately resulted in a categorical cross-entropy loss of 0.3040. Slightly lower classification accuracy was achieved for the testing dataset with 87.69% and a loss of 0.3398. Figure 12 shows different performance plots a) model loss vs epochs, b) model loss vs. time, and c) model accuracy vs epochs. The greatest drop of the loss takes place in the early stages of the training process as shown in Figure 12a) approximately epoch 10 and 12b) 750 seconds. Figure 12c) indicates an increasing trend of accuracy with increasing numbers of training runs (i.e., epochs). However, the positive return of increasing accuracy reduces with additional training runs.

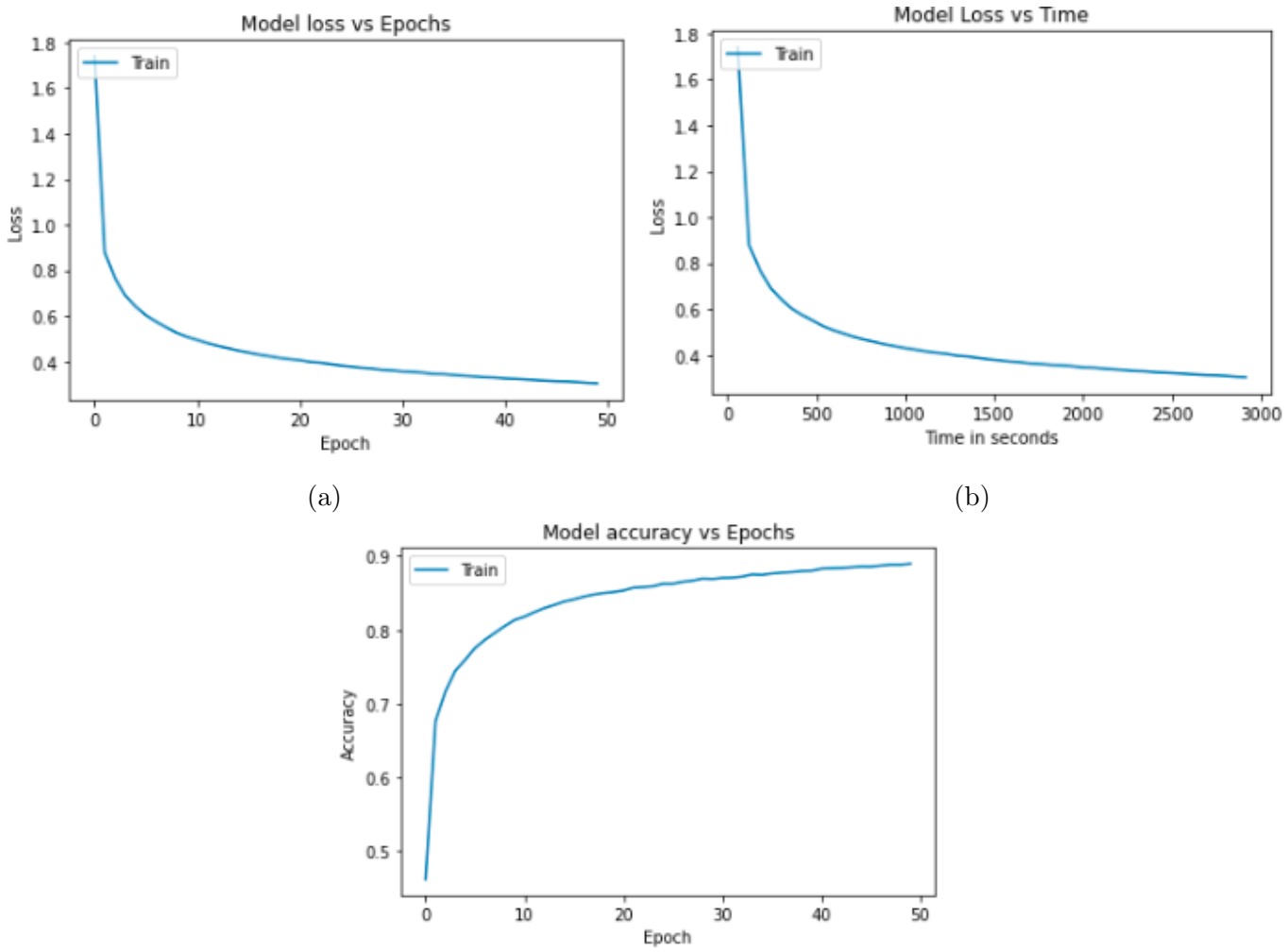


Figure 12: Performance plots for the big convolutional neural network: a) epoch-loss plot, b) time-loss plot, and c) epoch-accuracy plot.

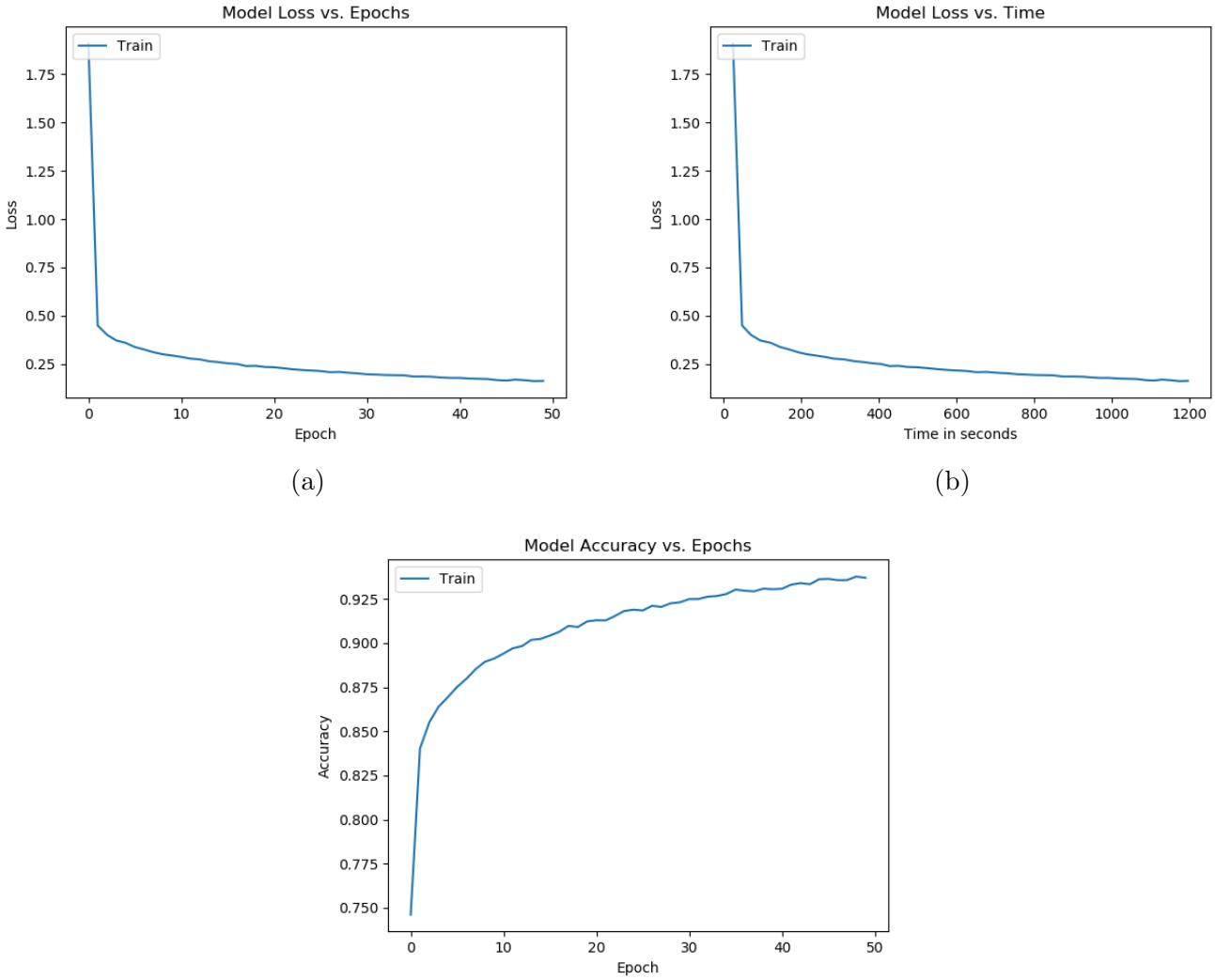
The confusion matrix (Figure 13) shows the classification performance of the big convolutional neural network for the test data set, where the rows represent the true class labels and the columns refer to the predicted labels. The classification accuracy for most classes lies above 85% (1000 samples per class), with the best performance for the class *trouser* with an accuracy of 97.4% and *bag* with an accuracy of 97.0%. The results further indicate, that specific types of clothes are more similar to each other than to other types of clothes. For example, the class *shirt* shows rather higher similarities between items from the class *T-shirt/top* (173 wrong assignments), *coat* (121), and *pullover* (94) as the wrongly assigned labels are high. Interestingly, the model can differentiate between classes related to any type of shoes (*ankle boot*, *sneaker*, and *sandals*) and all other classes, as the wrongly assigned labels, are mostly contributed to the shoe-related classes.

	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	871	2	18	16	10	3	67	0	13	0
Trouser	2	974	0	17	3	0	2	0	2	0
Pullover	15	2	805	9	114	0	48	0	7	0
Dress	23	9	15	875	49	0	25	0	4	0
Coat	1	1	61	22	840	0	69	0	6	0
Sandal	0	0	0	1	0	955	0	29	1	14
Shirt	173	2	94	22	121	0	568	0	20	0
Sneaker	0	0	0	0	0	14	0	962	1	23
Bag	4	1	6	2	6	3	4	4	970	0
Ankle boot	0	0	0	0	0	5	0	45	1	949

Figure 13: Confusion matrix for big convolutional neural network.

3.4 Task 4: Own Convolutional Neural Network

The convolutional neural network correctly classified 93.7% of the training images with a categorical cross-entropy loss of 0.1634. Slightly lower classification accuracy was achieved for the testing data with 89.89% and with a loss of 0.3177. Figure 14 shows different performance plots a) model loss vs. epochs, b) model loss vs. time, and c) model accuracy vs. epochs. The greatest drop of the loss function takes place in the early stages of the training process as shown in Figure 14a) and 14b). Figure 14c) shows how the accuracy increases as the numbers of training runs (i.e., epochs) increases. However, after epoch 45, the increase in accuracy slows down as it approaches its saturation point.



(a) (b)

(c)

Figure 14: Performance plots for the own modified convolutional neural network: a) epoch-loss plot, b) time-loss plot, and c) epoch-accuracy plot.

10-Class Confusion Matrix:

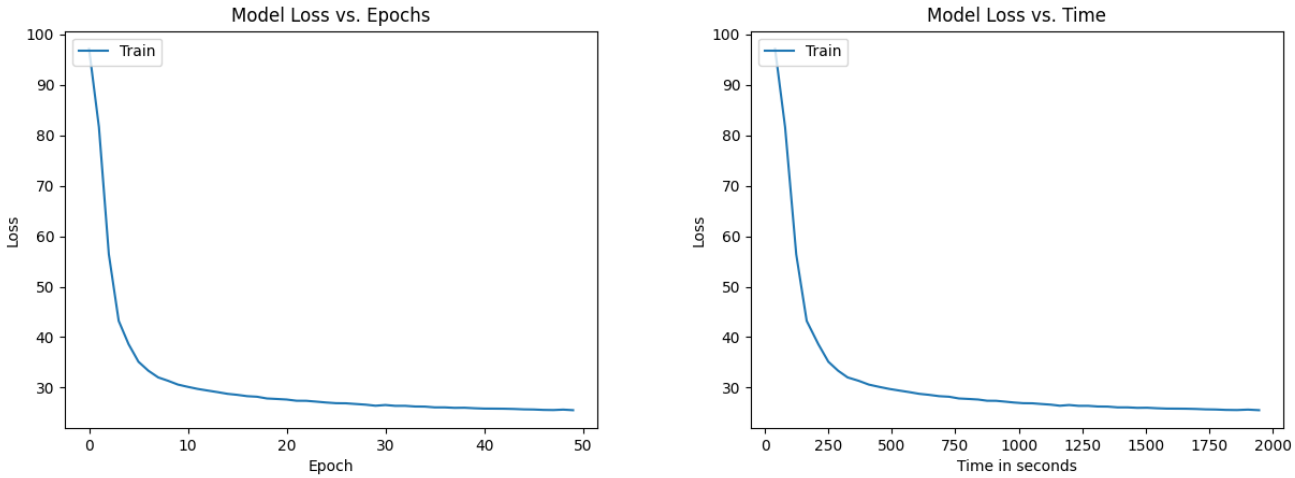
	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	831	0	20	14	3	1	125	0	6	0
Trouser	1	981	1	12	1	0	2	0	2	0
Pullover	10	0	829	5	102	0	53	0	1	0
Dress	10	7	13	908	32	0	26	0	4	0
Coat	1	1	68	36	855	0	38	0	1	0
Sandal	0	0	0	0	0	980	0	15	0	5
Shirt	98	1	73	22	107	0	696	0	3	0
Sneaker	0	0	0	0	0	11	0	974	0	15
Bag	2	1	2	6	3	1	6	5	974	0
Ankle boot	1	0	0	0	0	2	0	36	0	961

Figure 15: Confusion matrix for own modified convolutional neural network.

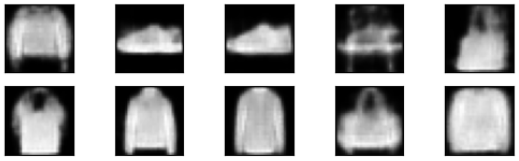
The confusion matrix (Figure 15) shows the classification performance of the convolutional neural network when predicting the test data, where the rows represent the true class labels and the columns the predicted labels. The classification accuracy for most classes lies above 80% (1000 samples per class), with the best performance for the class *Trouser* with an accuracy of 98.1%. The results also show that certain types of clothes are more similar to each other than to other types of clothes. For instance, the class *Shirt* shows higher similarities between items from the class *T-shirt/top* (98 misclassifications), *Pullover* (73 misclassifications), and *Coat* (107 misclassifications) as the misclassifications are higher. Like the deep CNN from task 3, the model can differentiate pretty well between the different types of shoes (*Ankle boot*, *Sneaker*).

3.5 Task 5: Variational Autoencoder

The variational autoencoder with latent dimension of 10 achieved a training loss of 26.05 and test loss of 26.66. Figure 16 shows different performance plots a) model loss vs. epochs, b) model loss vs. time, and c) set of clothes generated by randomly choosing 10 latent vectors. The greatest drop of the loss function occurs in the early stages of the training process and decreases as the number of epochs and time increases as shown in Figure 16a) and 16b). Figure 16c) shows how the decoder model was able to transform random samples from the 10-dimensional latent space into images that indeed look similar to one of the fashion items from the Fashion-MNIST dataset.



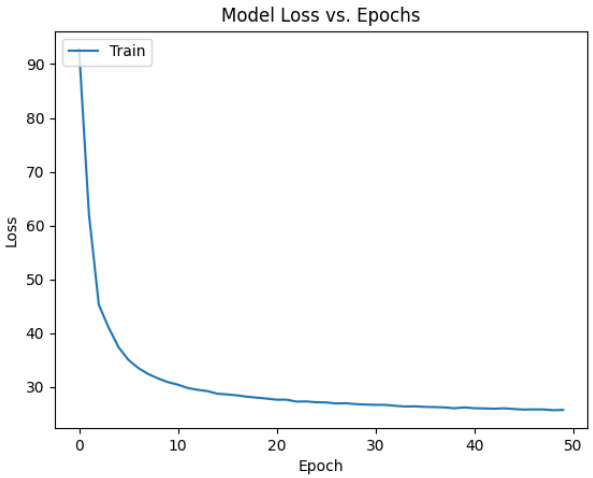
(a) (b)



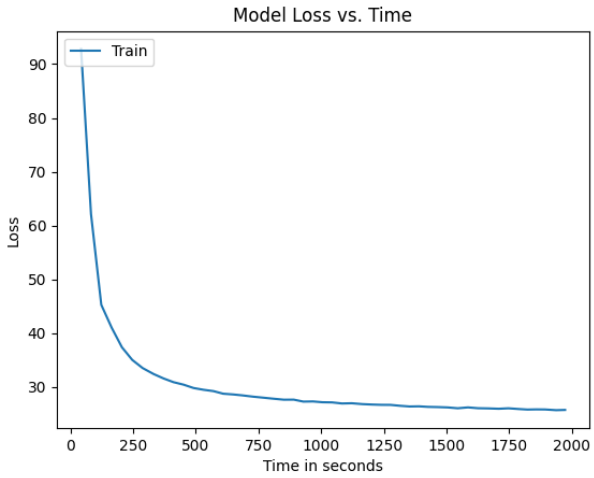
(c)

Figure 16: Performance plots for the variational autoencoder with latent dimension 10: a) epoch-loss plot, b) time-loss plot, and c) set of clothes generated by randomly choosing 10 latent vectors and presenting the resulting images.

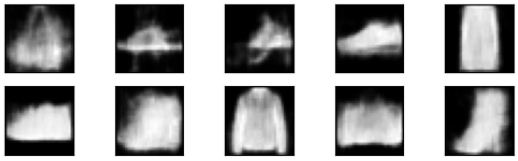
The variational autoencoder with latent dimension of 32 achieved a training loss of 25.3 and test loss of 25.61. Figure 17 shows different performance plots a) model loss vs. epochs, b) model loss vs. time, and c) set of clothes generated by randomly choosing 10 latent vectors. Again, the greatest drop of the loss function occurs in the early stages of the training process and decreases as the number of epochs and time increases as shown in Figure 17a) and 17b). Figure 17c) shows how the decoder model was able to transform random samples from the 32-dimensional latent space into images that picture one of the fashion items from the Fashion-MNIST dataset.



(a)



(b)



(c)

Figure 17: Performance plots for the variational autoencoder with latent dimension 32: a) epoch-loss plot, b) time-loss plot, and c) set of clothes generated by randomly choosing 10 latent vectors and presenting the resulting images.

4 Conclusion

Table 1 gives an overview of the performance accuracy and respective loss for either the train or test data for each model. The highest accuracy was achieved by the CNN of task 4 with a training accuracy of 93.7% and a test accuracy of 89.9%. When comparing the two variational autoencoders, the model with latent dimension 32 did not perform significantly better than the model with latent dimension 10 which can also been seen by Figure 16c) and 17c).

Table 1: Output summary, showing accuracy and loss for final training epoch and test dataset.

Model	Training Data (Final Epoch)		Test Data	
	Loss	Accuracy	Loss	Accuracy
Fully Connected NN	0.3897	0.8617	0.4163	0.8461
Small CNN	0.2865	0.8974	0.3225	0.8859
Big CNN	0.3040	0.8895	0.3398	0.8769
Own CNN	0.1634	0.9370	0.3177	0.8989
VAE (latent dim = 10)	26.05	NA	26.66	NA
VAE (latent dim = 32)	25.3	NA	25.61	NA

5 Running the Code

The file proj3_task1234.py performs tasks 1-4. The `main` method takes the command line variables: `task1`, `task2`, `task3`, and `task4`. To run tasks 1-4 use the following example command:

```
python3 proj3_task1234.py "task1"
```

The file proj3_task5.py performs task 5. The `main` method takes the command line variables: `task5_vae1` (VAE with latent dimension 10) and `task5_vae2` (VAE with latent dimension 32). To run task 5 use the following example command:

```
python3 proj3_task5.py "task5_vae1"
```